

# **Session 3**

## **White-Box Testing**

**xfzhang@suda.edu.cn**

## Objectives

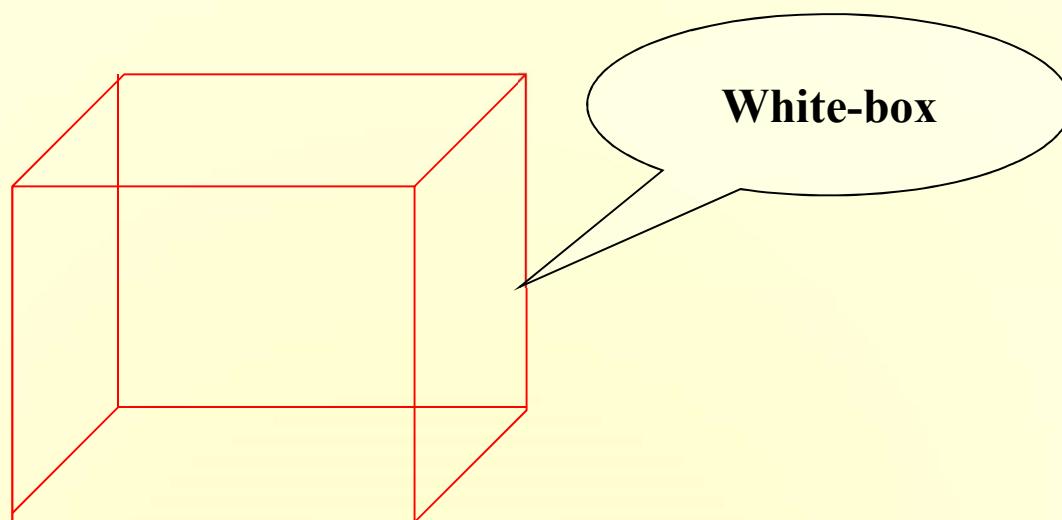
- ◆ Review of Session 2
  - ◆ Comparison of some concepts
    - ◆ What is “defect”?
    - ◆ Verification / Validation
    - ◆ Testing / Debugging
    - ◆ White-box testing / Black-box testing
  - ◆ Testing life cycle
  - ◆ Model of testing
  - ◆ Types of testing

## Objectives

- ◆ In this session, you will learn
  - ◆ Basic Concepts of White box testing
  - ◆ Logic Coverage
  - ◆ Control Flow Graph
  - ◆ Basis Path Testing
  - ◆ Loop Testing
  - ◆ Data Flow Testing

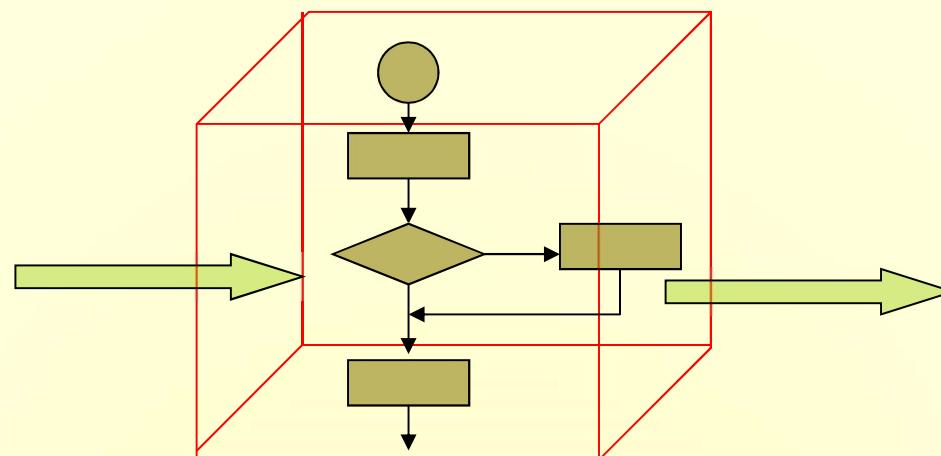
# 3.1 Basic Concepts

- \* White-box testing
  - \* It is a validation technique software engineers can use to examine if their code works as expected.
  - \* It is also known as structural testing, clear box testing, and glass box testing.



# 3.1 Basic Concepts

- \* White-box testing
  - \* It indicates that you have full visibility of the internal workings of the software product, specifically, **the logic and the structure** of the code.



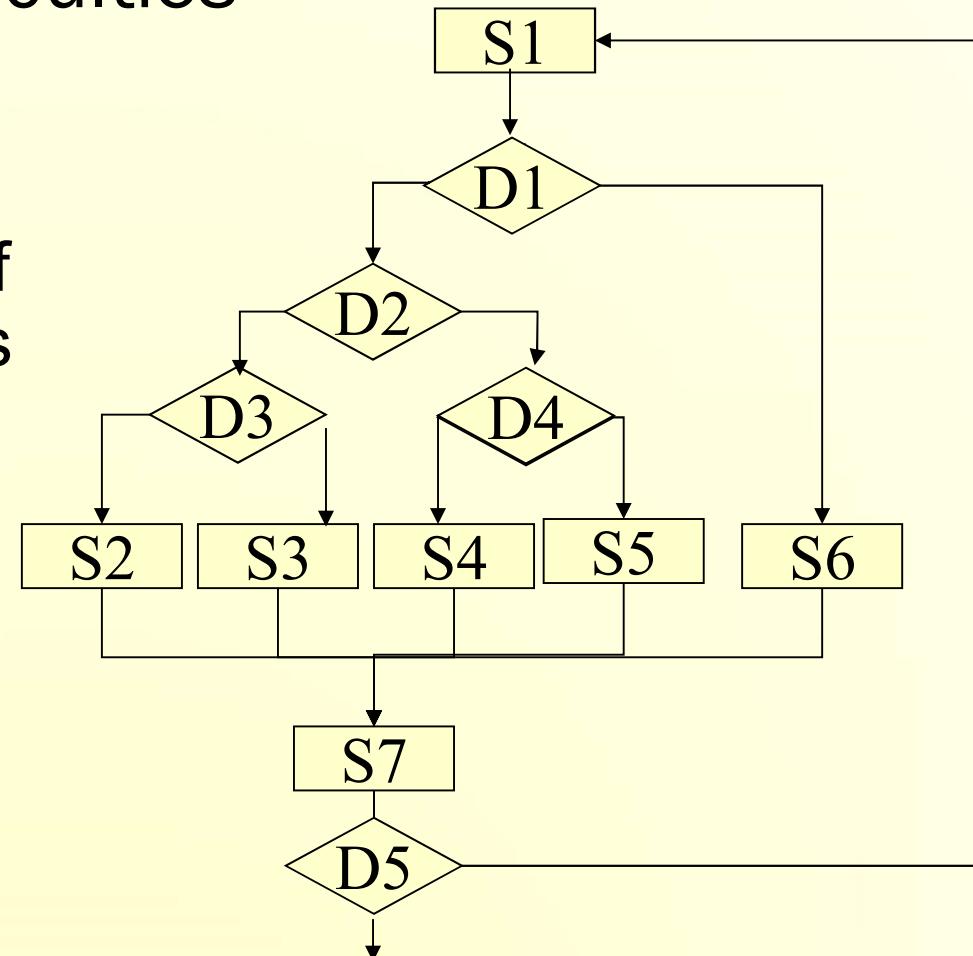
# 3.1 Basic Concepts

	<b>White-box Testing</b>	<b>Black-box Testing</b>
程序结构	已知程序结构	未知程序结构
规模	小规模测试	大规模测试
依据	详细设计说明、源代码	需求说明、概要设计说明
面向	程序结构	输入输出接口/功能要求
适用	单元测试	集成、系统、验收测试
测试人员	开发人员	专门测试人员/外部人员(用户)
优点	能够对程序内部的特定部位进行覆盖	能站在用户的立场上进行测试
缺点	无法检验程序的外部特性,不能检测对需求的遗漏	不能测试程序内部特定部位,如果规格说明有误,仍无法发现

# 3.1 Basic Concepts

## \* White-box Testing Difficulties

- For multiple choices and nesting cycle of the procedure, the number of different possible paths is astronomical.
- cycle $\leq$ 20times
- Different paths is  $5^{20}$ , if the implementation time of each path is 1 ms , 3024years are needed.



## 3.1 Basic Concepts

- \* Why we can't use exhaustive testing?
  - \* Time consuming is expensive.
  - \* Path exhaustive testing can't detect the wrong because of **path omission**.
  - \* Path exhaustive testing can not discovery some errors **associated with the data**.(eg. boundary, specific input)

## 3.1 Basic Concepts

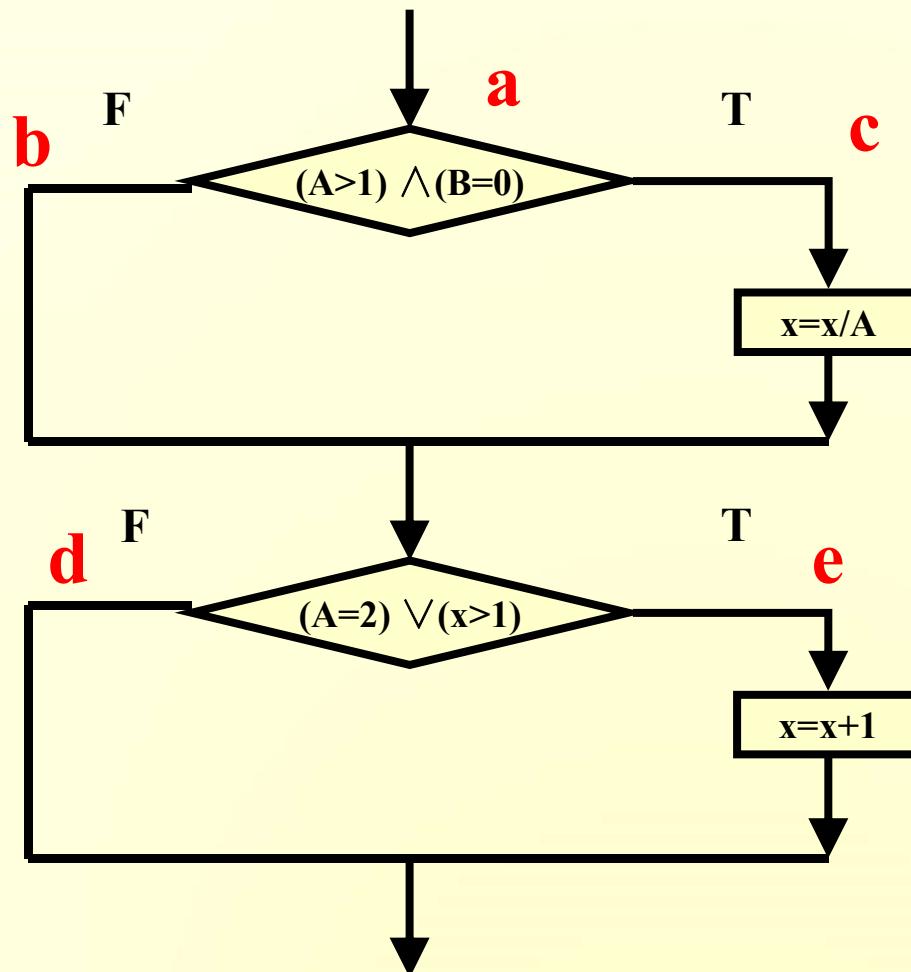
- \* White-box testing must follow several principles:
  - \* All *independent* path in a module must be implemented at least once. (**Basis path testing**)
  - \* All logic values require two test cases: true and false. (**Logic coverage**)
  - \* Inspection procedures of internal data structure, and ensuring the effectiveness of its structure.  
**(Static Testing + Data Flow Testing)**
  - \* Run all cycles within operational range. (**Loop testing**)

## 3.2 Logic Coverage

- \* Logic coverage
  - \* Statement Coverage
  - \* Decision Coverage
  - \* Condition Coverage
  - \* Condition-Decision Coverage
  - \* Condition Combination Coverage
  - \* Path Coverage
  - \* Complete Coverage
  - \* Modified Condition/Decision Coverage

## 3.2 Logic Coverage

- \* Flow Chart of Sample Code    A,B,X: INT



Path:

L1(a → c → e)

L2(a → b → d)

L3(a → b → e)

L4(a → c → d)

## 3.2 Logic Coverage—Statement Coverage

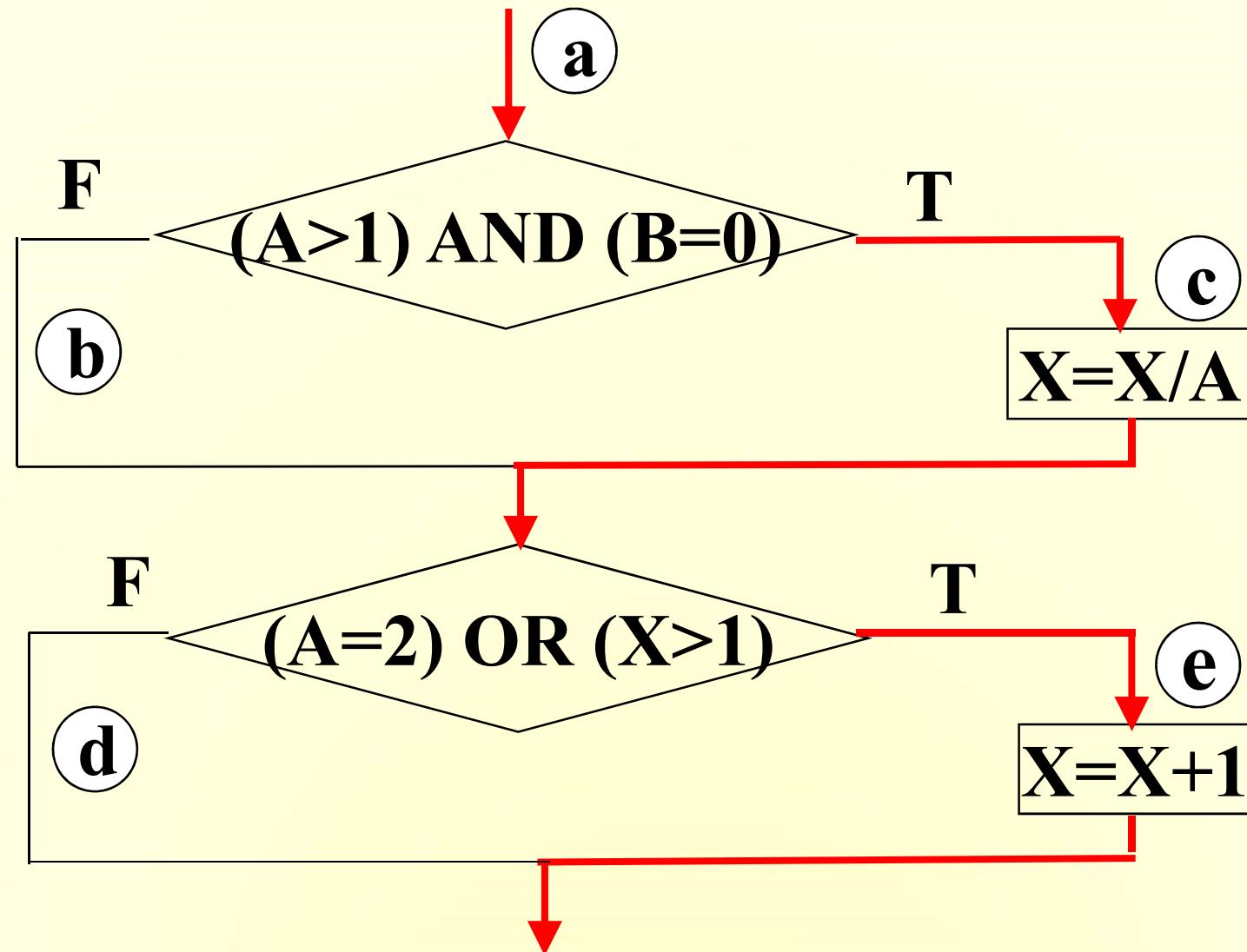
- \* Statement coverage (语句覆盖)
  - \* Statement coverage is to design a number of test cases, making each executable statement implement at least once.
  - \* In diagram, **all the executable statements are in the path L1**, so choose path L1 to design test case

## 3.2 Logic Coverage—Statement Coverage

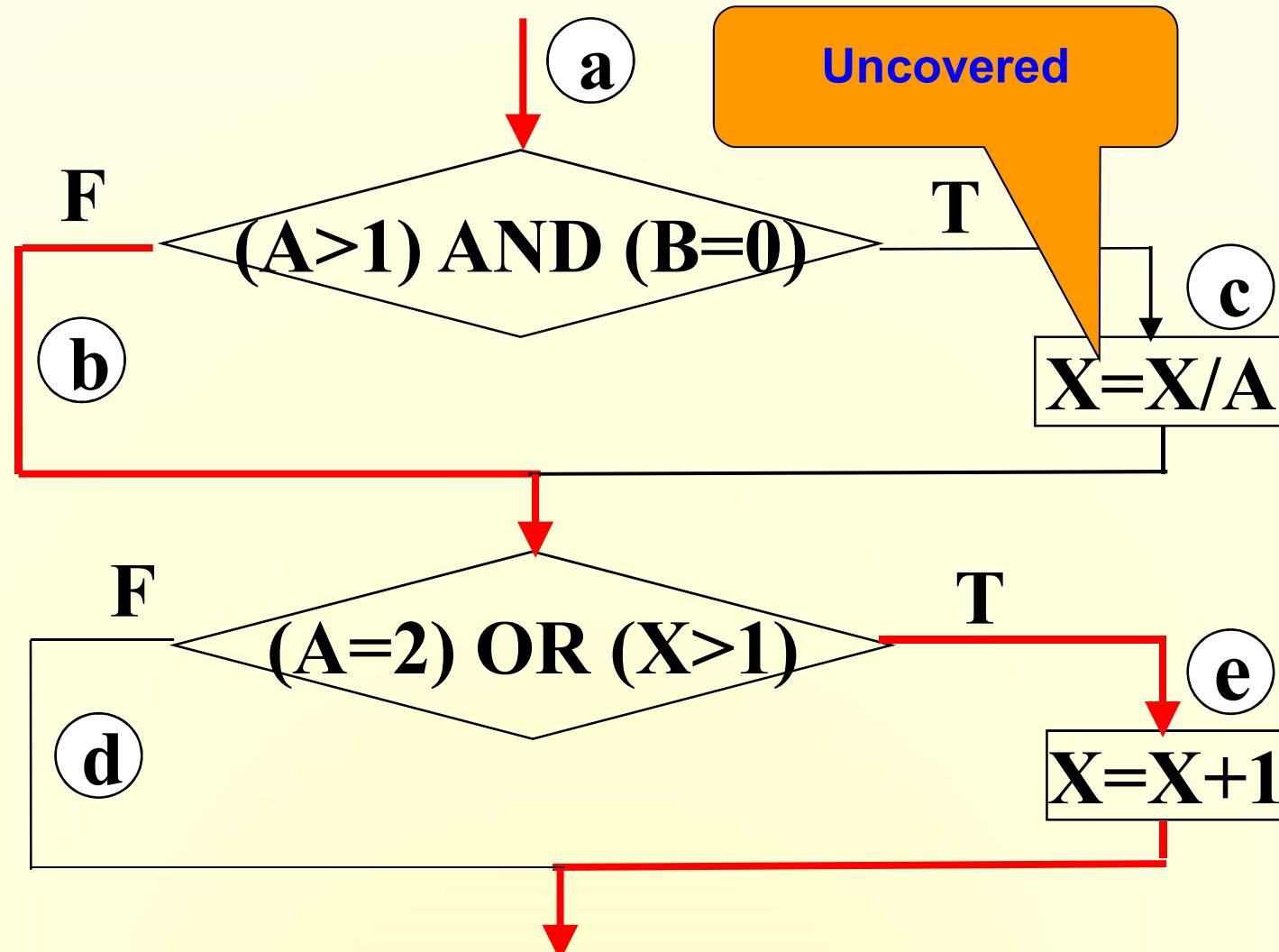
L1:[(A=2) and (B=0)] or [(A>1) and (B=0) and (x/A>1)]

TC	A	B	X	Path	
Case1	2	0	3	ace	✓
Case2	2	1	3	abe	✗

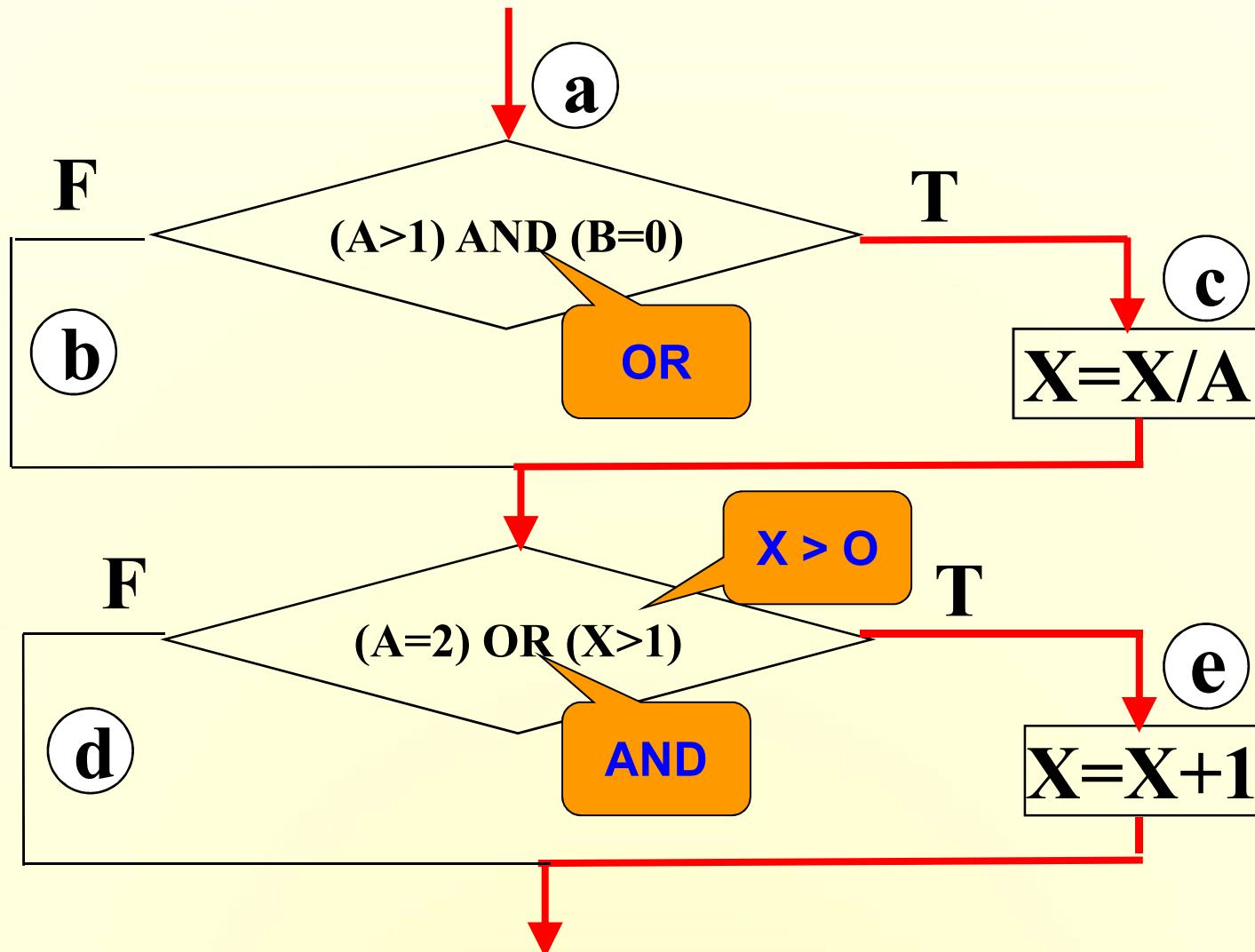
# Case1: $A=2$ , $B=0$ , $X=3$



## Case2: $A=2$ , $B=1$ , $X=3$



# Case1: A=2, B=0, X=3

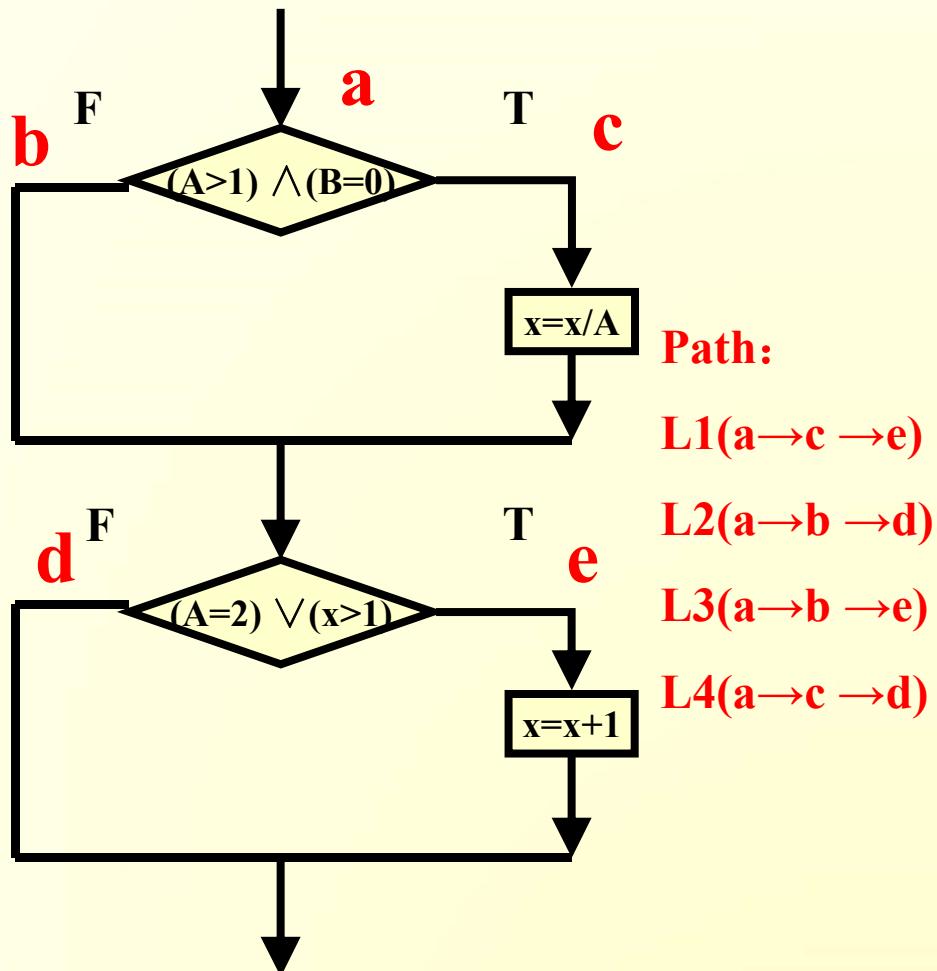


**Statement coverage is weakest**

## 3.2 Logic Coverage—Decision coverage

- \* Decision Coverage (判定覆盖) is to design a number of test cases, make the true and false branches of each judgment may go through at least once.
- \* “Switch–Case” – each branches

## 3.2 Logic Coverage—Decision coverage



**(2, 0, 3) covers ace [L1]**  
**(1, 1, 1) covers abd [L2]**

or

**L1(a → c → e)**  
**L2(a → b → d)**  
**L3(a → b → e)**  
**L4(a → c → d)**

**(2, 1, 1) covers abe [L3]**  
**(3, 0, 3) covers acd [L4]**

## 3.2 Logic Coverage—Decision coverage

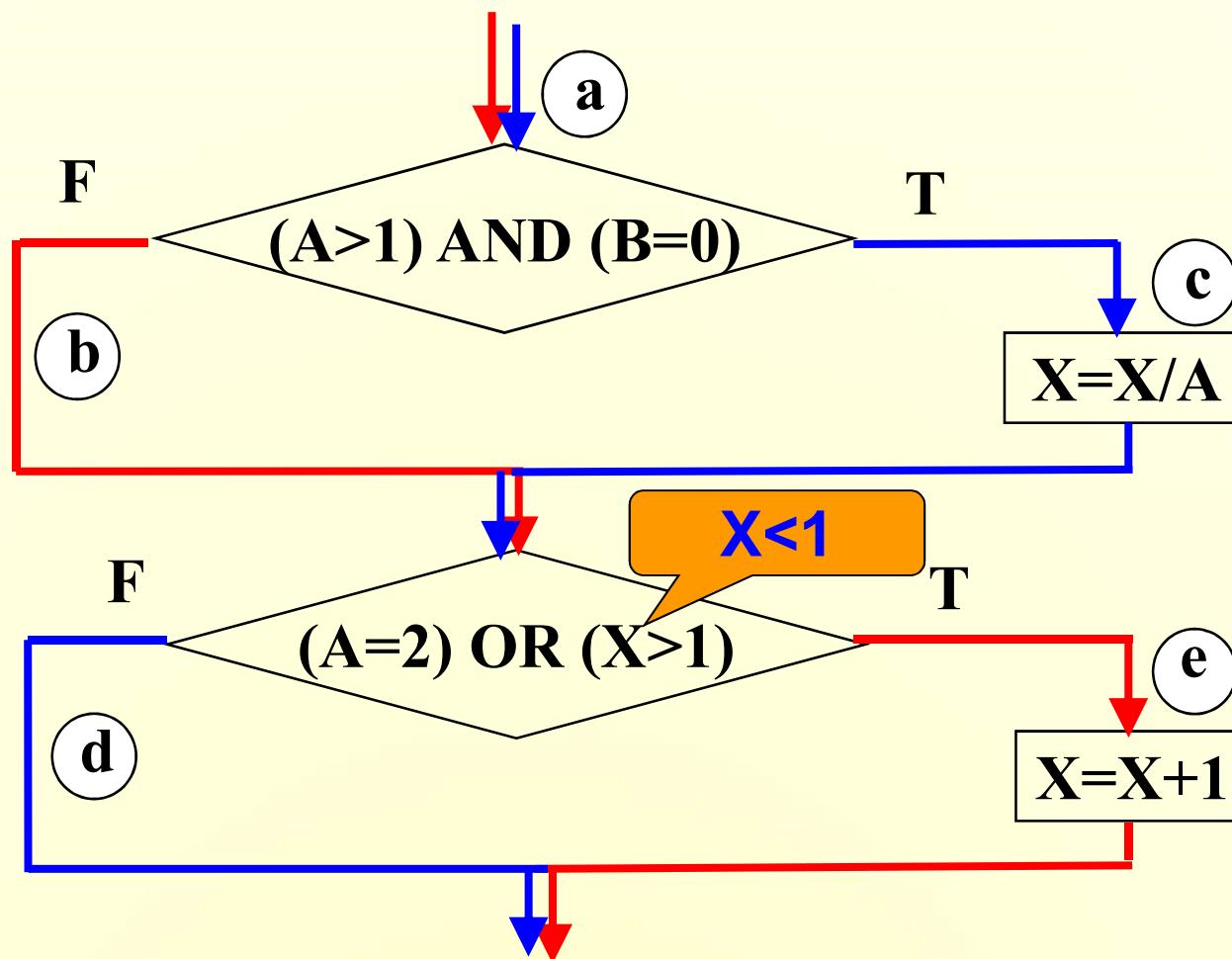
- \* Please consider that
  - \* If  $x > 1$  is written wrong as  $x < 1$ 
    - \* t1 (2, 0, 3)   t2 (1, 1, 1)
    - \* t3 (2, 1, 1)   t4 (3, 0, 3)

**Decision Coverage is not guaranteed they can detect the wrong conditions in judgment**

Eg.

Case3: A=2, B=1, X=1

Case4: A=3, B=0, X=3

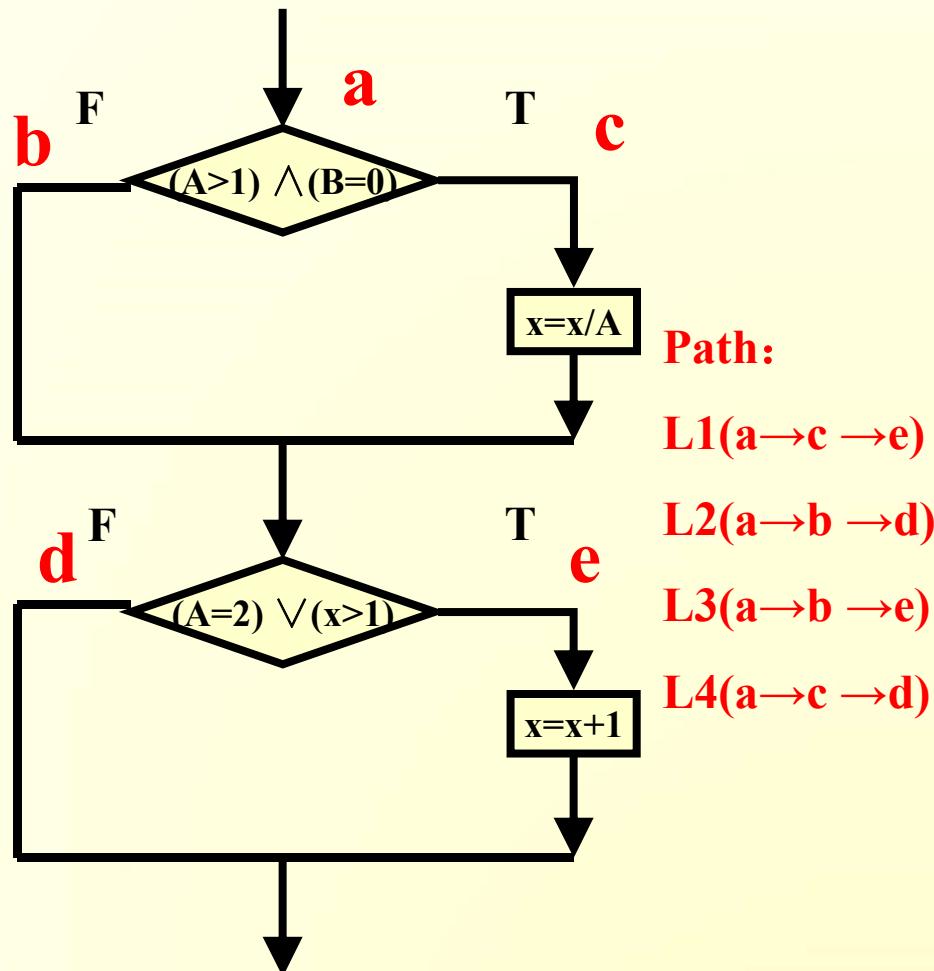


Decision coverage is not strong enough.

## 3.2 Logic Coverage—Condition coverage

- \* Conditions coverage (条件覆盖) is to design a number of test cases, make **possible values of each condition** in the procedure may implement at least once.
- \* Decision vs. condition

## 3.2 Logic Coverage—Condition coverage



For the first judgment

Condition  $A > 1$  true value is T1, false value is !T1

Condition  $B = 0$  true value is T2, false value is !T2

For the second judgment

Condition  $A = 2$  true value is T3, false value is !T3

Condition  $X > 1$  true value is T4, false value is !T4

## 3.2 Logic Coverage—Condition Coverage

Test case
(2, 0, 4)
(1, 1, 1)

Test case
(1, 0, 3)
(2, 1, 1)

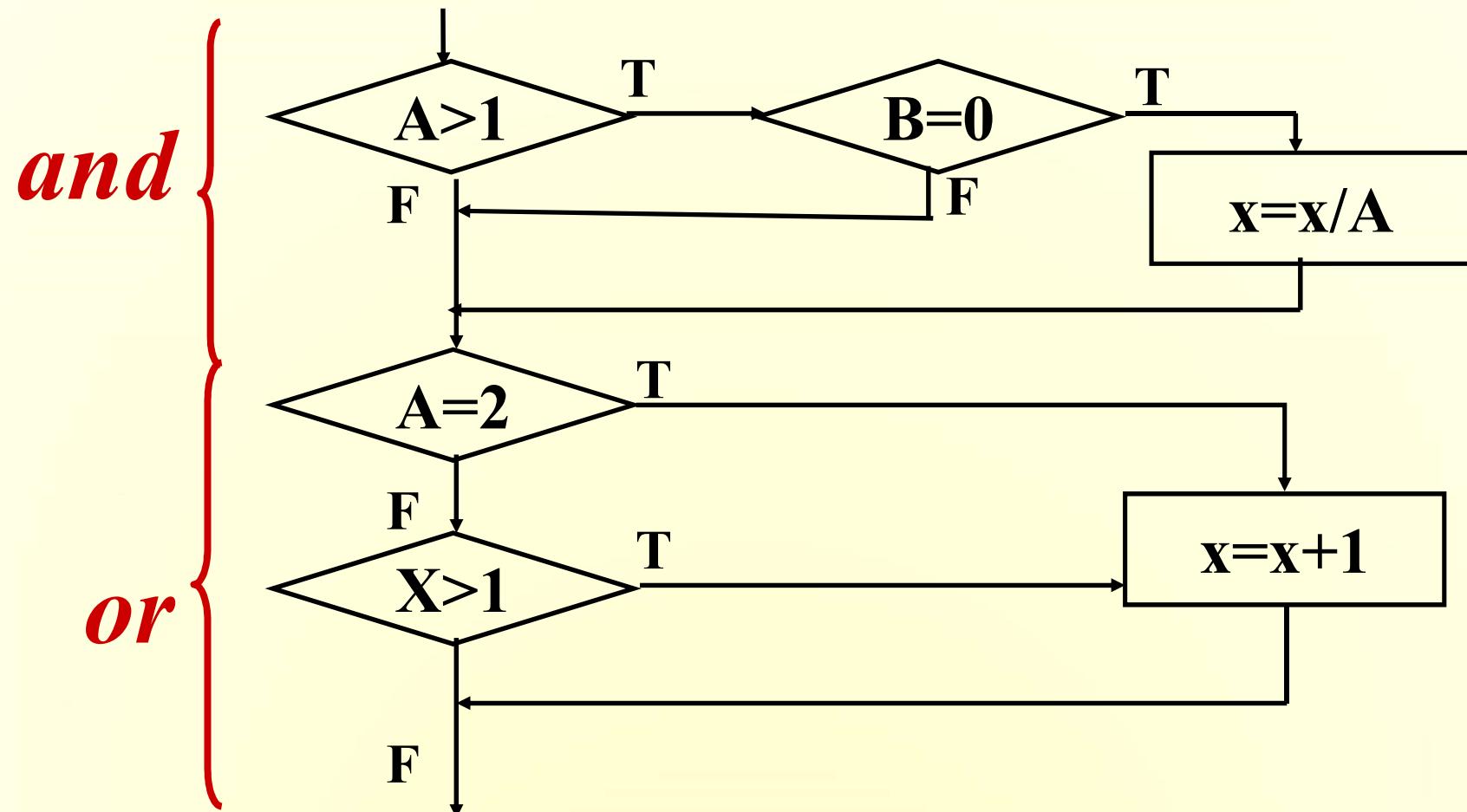
Test case
(2, 0, 4)
(3, 1, 1)

## 3.2 Logic Coverage—Condition / Decision Coverage

- \* Condition/decision coverage (条件判断覆盖) is to design sufficient test cases, make **all possible conditions** of each judgment implement at least once, and make **all possible results of each judgment** implement at least once.
- \* An example to meet this requirement.

Test case	Path	Condition value	Coverage branch
(2, 0, 4)	ace(L1)	T1 T2 T3 T4	c, e (T, T)
(1, 1, 1)	abd(L2)	!T1 !T2 !T3 !T4	b, d (F, F)

## 3.2 Logic Coverage—Decomposition



## 3.2 Logic Coverage—Condition Combination Coverage

- \* Condition combination coverage (条件组合覆盖) is to design sufficient test cases, make **all possible condition combinations** of each judgment implement as least once.
- \* If the test cases meet the condition combination coverage, then they certainly meet the decision coverage, condition coverage and condition/decision coverage.

## 3.2 Logic Coverage—Condition Combination Coverage

- ①  $A > 1, B = 0$  as T1T2
  - ②  $A > 1, B \neq 0$  as T1!T2
  - ③  $A \geq 1, B = 0$  as !T1T2
  - ④  $A \geq 1, B \neq 0$  as !T1!T2
- 
- ⑤  $A = 2, X > 1$  as T3T4
  - ⑥  $A = 2, X \geq 1$  as T3!T4
  - ⑦  $A \neq 2, X > 1$  as !T3T4
  - ⑧  $A \neq 2, X \geq 1$  as !T3!T4

## 3.2 Logic Coverage—Condition Combination Coverage

Test case	Path	Coverage condition	Combination coverage No.
(2, 0, 4)	ace(L1)	T1 T2 T3 T4	① ⑤
(2, 1, 1)	abe(L3)	T1 !T2 T3 !T4	② ⑥
(1, 0, 3)	abe(L3)	!T1 T2 !T3 T4	③ ⑦
(1, 1, 1)	abd(L2)	!T1 !T2 !T3 !T4	④ ⑧

**There are four paths altogether in the procedure.  
Although the four test cases above cover all condition  
combinations and 4 branches, only 3 paths are covered  
and the path “acd” is missed.**

## 3.2 Logic Coverage—Path Coverage

- \* Path coverage (路径覆盖) is to design enough test case to cover all possible path in the procedure.

No.	Test case	path	Coverage condition
1	(2, 0, 4)	ace(L1)	T1T2T3T4
2	(1, 1, 1)	abd(L2)	!T1!T2!T3!T4
3	(2, 1, 1)	abe(L3)	T1 !T2 T3 !T4
4	(3, 0, 3)	acd(L4)	T1T2!T3!T4

## 3.2 Logic Coverage—Complete Coverage

- None of aforementioned coverage strategies can cover all test cases. We consider using the combination of them to achieve better coverage.
- Complete Coverage (全覆盖) = Condition Combination Coverage + Path Coverage

# Test cases satisfy both condition combination coverage and path coverage

A	B	X	Path	Combination Covered	Condition Covered
2	0	3	a c e	① ⑤	T1 T2 T3 T4
2	1	1	a b e	② ⑥	T1 F2 T3 F4
1	0	3	a b e	③ ⑦	F1 T2 F3 T4
1	1	1	a b d	④ ⑧	F1 F2 F3 F4
3	0	3	a c d	① ⑧	T1 T2 F3 F4

## \* Exercise1

---

```
int Func_Check(bool a, bool b, bool c)
```

```
{
```

```
    int x;
```

```
    x = 0;
```

```
    if (a&&b&&c)
```

```
        x =1;
```

```
    return x;
```

```
}
```

which one can achieve statement coverage? ( )

- A. a = F, b = F, c = F
- B. a = T, b = T, c = F
- C. a = T, b = T, c = T
- D. a = T, b = F, c = T

\* Exercise2

```
int Fun_Check(bool a, bool b, bool c)
{
```

```
    int x;
    x = 0;
    if (a| |b| |c)
        x =1;
```

```
    else
        x = 0;
    return x;
```

```
}
```

which one can achieve decision coverage? ( )

- A. a = T, b = T, c = F ; a = T, b =F, c = F
- B. a = T, b = F, c = F; a = F, b = T, c = F
- C. a = F, b = T, c = T; a = F, b = F, c = T
- D. a = F, b = F, c = T; a = F, b = F, c = F

\* Exercise3

```
int Func_Check (int a, int b, int c)
{
    int x;
    x = 0;
    if ((a==2)&&(b>2))||(c<0))
        x =1;
    else
        x = 0;
    return x;
}
```

which one can achieve condition/decision coverage? ( )

- A. a =4, b = 7, c = 8; a = 7, b = 5, c = 4
- B. a =2, b = 3, c = -2; a = 7, b = 5, c = 5
- C. a =4, b = 3, c = 8; a = 7, b = 1, c = 4
- D. a =2, b = 3, c = -2; a = 0, b = 1, c = 5

## \* Exercise4

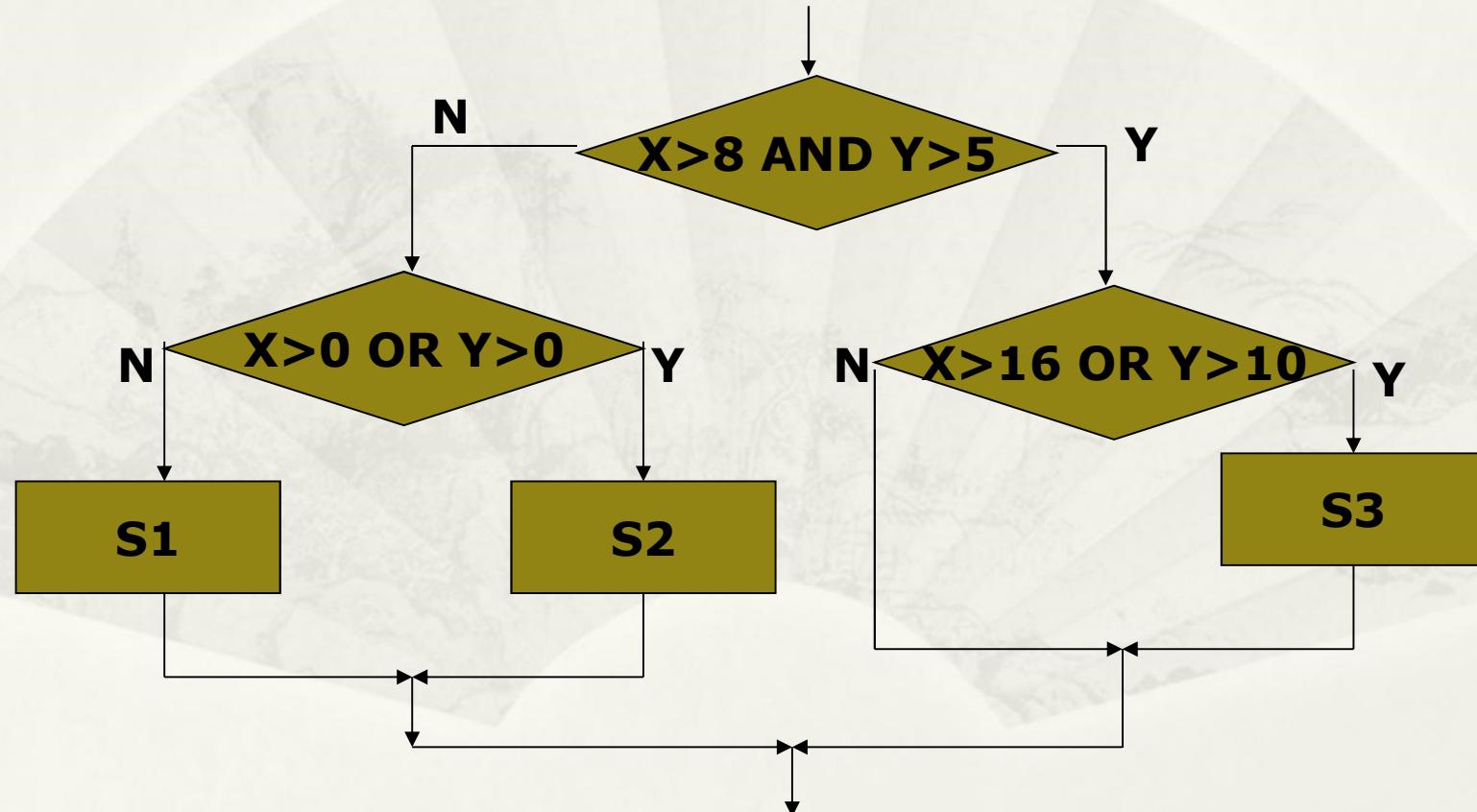
```
int Score (int ChineseScore, int MathScore, int PhysicalScore)
{
    int x;
    x = 0;
    if ((ChineseScore>80)&&( MathScore>=80)&&( PhysicalScore>=80))
        x =1;
    else
        x = 0;
    return x;
}
```

which one can achieve condition/decision coverage? ( )

- A. ChineseScore = 70, MathScore = 90, PhysicalScore = 95  
    ChineseScore = 70, MathScore = 70, PhysicalScore = 70
- B. ChineseScore = 81, MathScore = 90, PhysicalScore = 82  
    ChineseScore = 95, MathScore = 90, PhysicalScore = 93
- C. ChineseScore =100, MathScore = 90, PhysicalScore = 82  
    ChineseScore =60, MathScore = 35, PhysicalScore = 44
- D. ChineseScore =45, MathScore = 67, PhysicalScore =56  
    ChineseScore =64, MathScore = 78, PhysicalScore = 77

## Exercise 5

- \* Design test cases for the following flowchart to satisfy 7 types of coverage. (Note: X and Y are signed integers or 0)

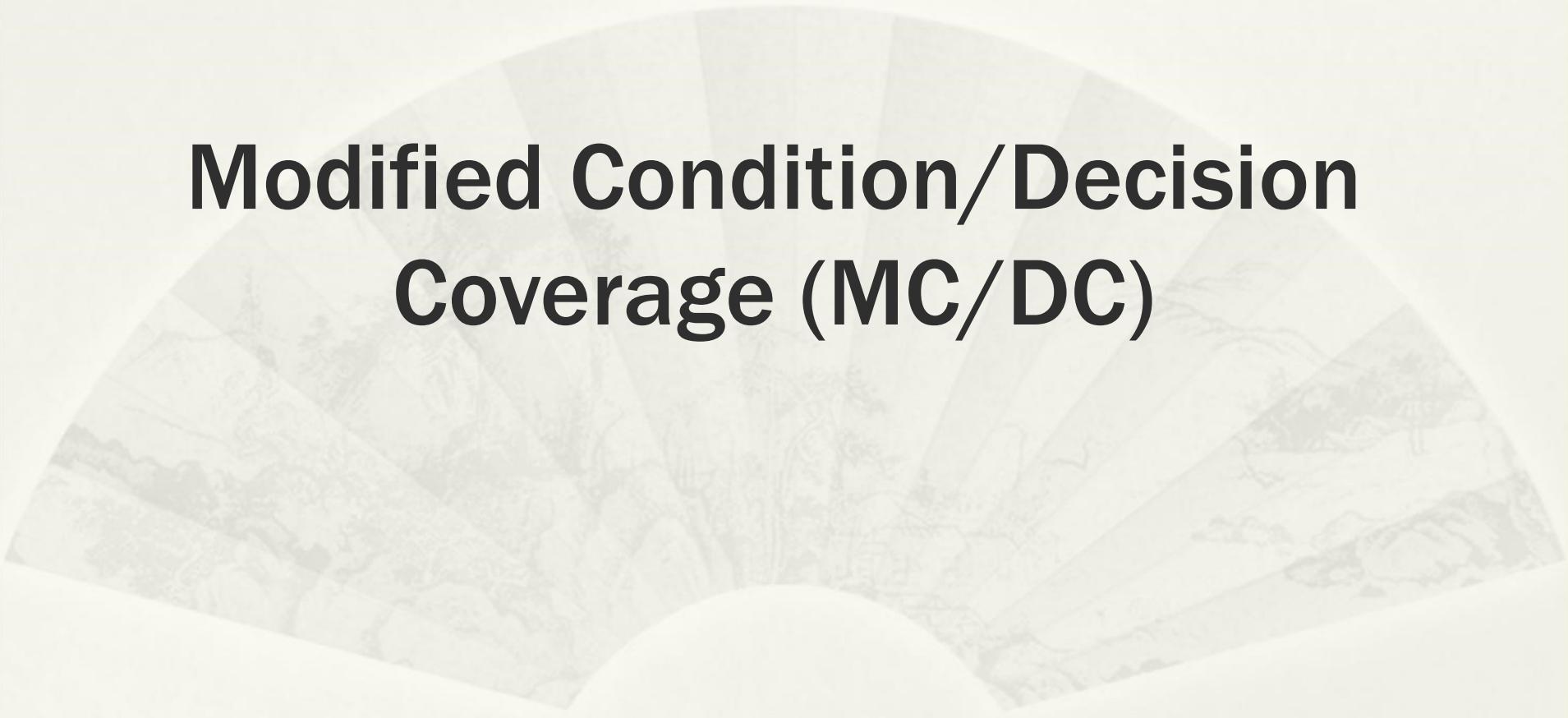


## Exercise 6

- \* Try to analyze the relationship among 7 different coverage strategies
  - \* Statement Coverage
  - \* Decision Coverage
  - \* Condition Coverage
  - \* Condition-Decision Coverage
  - \* Condition Combination Coverage
  - \* Path Coverage
  - \* Complete Coverage

---

# **Modified Condition/Decision Coverage (MC/DC)**



# MC/DC

---

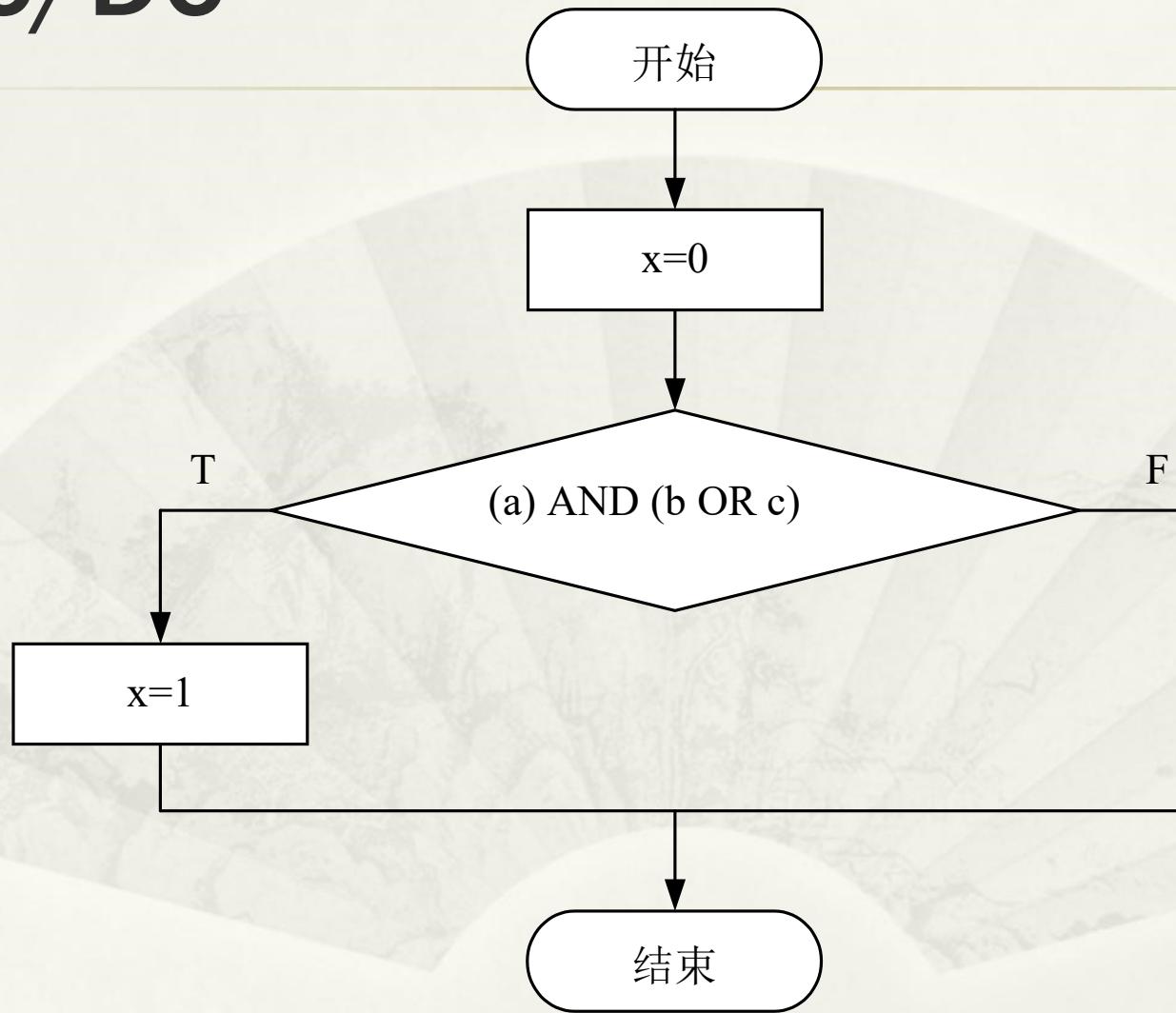
- \* A code coverage criterion that requires:
  - \* Each entry and exit point is invoked
  - \* Each decision takes every possible outcome
  - \* Each condition in a decision takes every possible outcome
  - \* Each condition in a decision is shown to **independently affect** the outcome of the decision.

# MC/DC

---

- \* Independence of a condition is shown by proving that only one condition changes at a time.
- \* MC/DC is used in avionics software development guidance DO-178B and DO-178C to ensure adequate testing of the most critical (Level A) software.

# MC/DC



```
int function1 (bool a, bool b, bool c)
{
    int x=0;
    if(a&&(b||c))
        x=1;
    return x;
}
```

---

- \* Exercise

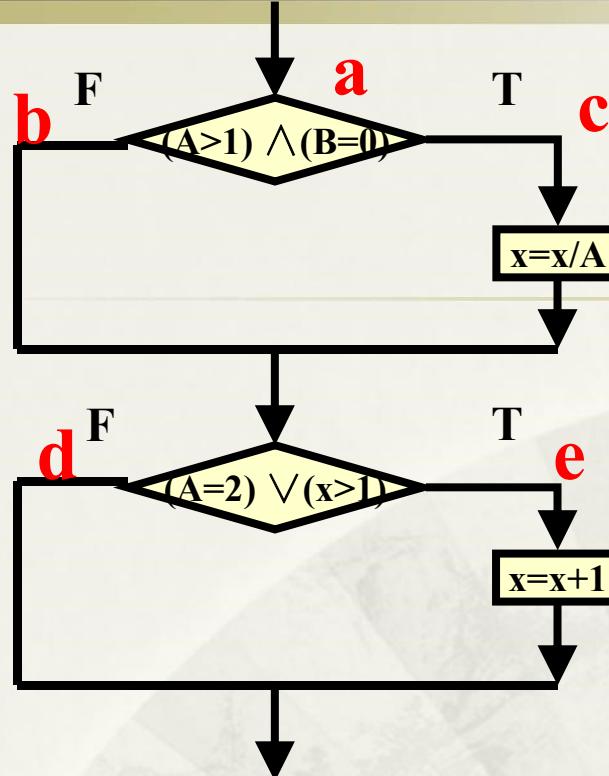
- \* Decision coverage
- \* Condition coverage
- \* Condition combination coverage

# MC/DC

序号	a	b	c	a&&(b  c)	a	b	c
1	T	T	T	T	5		
2	T	T	F	T	6	4	
3	T	F	T	T	7		4
4	T	F	F	F		2	3
5	F	T	T	F	1		
6	F	T	F	F	2		
7	F	F	T	F	3		
8	F	F	F	F			

$n+1 \sim 2^n$

$\{1, 2, 3, 4, 5\} / \{2, 3, 4, 6\}$



**Path:**

- L1(a → c → e)
- L2(a → b → d)
- L3(a → b → e)
- L4(a → c → d)

- \* Design test cases to satisfy MC/DC of this example.

Test case	Path	Coverage condition	Combination coverage No.
(2, 0, 4)	ace(L1)	T1 T2 T3 T4	① ⑤
(2, 1, 1)	abe(L3)	T1 !T2 T3 !T4	② ⑥
(1, 0, 3)	abe(L3)	!T1 T2 !T3 T4	③ ⑦
(1, 1, 1)	abd(L2)	!T1 !T2 !T3 !T4	④ ⑧

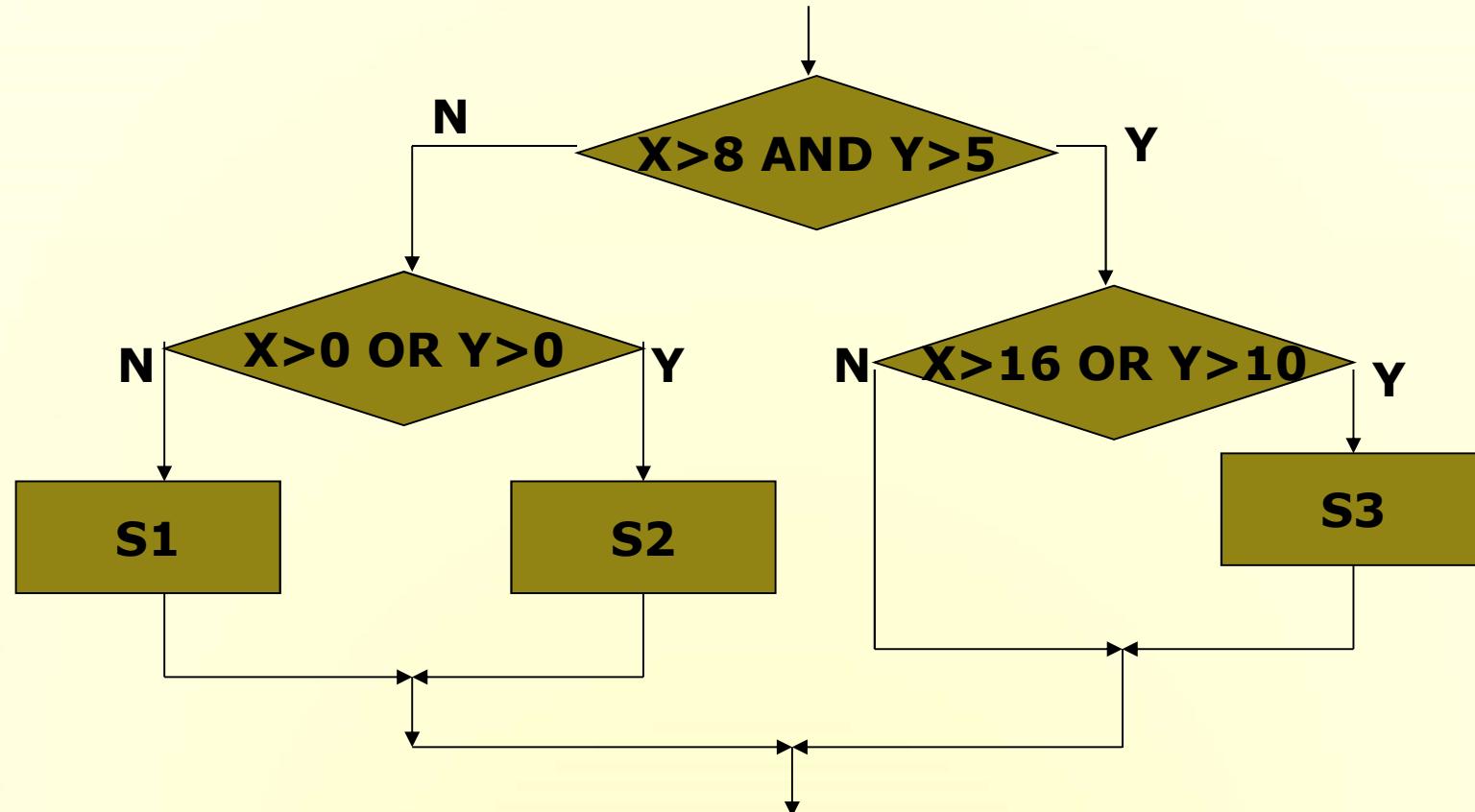
# MC/DC

---

- \* The MC/DC criterion is much stronger than the condition/decision coverage.
- \* The MC/DC criterion is NOT stronger than Condition Combination Coverage.
- \* Update Exercise 5/6

## Exercise 5'

- \* Design test cases for the following flowchart to satisfy 8 types of coverage. (Note: X and Y are signed integers or 0)



## Exercise 6'

- \* Try to analyze the relationship among 8 different coverage strategies
  - \* Statement Coverage
  - \* Decision Coverage
  - \* Condition Coverage
  - \* Condition-Decision Coverage
  - \* Condition Combination Coverage
  - \* Path Coverage
  - \* Complete Coverage
  - \* Modified Condition/Decision Coverage